SENSOR NETWORK MIDDLEWARE FOR MANAGING A CROSS-LAYER ARCHITECTURE *

Christophe J. Merlin
Department of Electrical and Computer Engineering
University of Rochester
Rochester, NY, USA
merlin@ece.rochester.edu

Wendi B. Heinzelman
Department of Electrical and Computer Engineering
University of Rochester
Rochester, NY, USA
wheinzel@ece.rochester.edu

Abstract Cross-layer designs have received much attention recently. While not as general as layered architectures, they prove to be more tunable and energy-efficient in many scenarios. This flexibility can be exploited by a middleware whose principal task is to adapt quality of service provided by the network to the application's needs using the pre-defined parameters of the cross-layer protocol. In this paper, we study the ways in which a middleware (MiLAN) can control a cross-layer protocol for wireless sensor networks (DAPR), thereby ensuring that the network provides the application's required quality of service while removing this burden from the application designer.

Keywords: Cross-layer protocols, middleware, wireless sensor networks

1. Introduction

The sensor network community has dedicated much attention to cross-layer protocols in recent years. In order to increase the effectiveness of cross-layer architectures for different applications, oftentimes they have several parameters whose value can be tuned to better serve each specific application. However, it would not be reasonable to expect the applica-

*This work was supported in part by NSF #CNS-0448046.

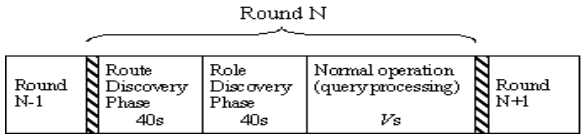

Figure 1. A DAPR round.

tion itself to control these different parameters—this is better handled through an intelligent middleware. The middleware's task is to coordinate application needs to quality of service rendered by the network, using commands understandable by the protocol. Our goal in this paper is to show how a sensor network middleware can meet this requirement by appropriately managing a cross-layer sensor network protocol.

We base our work on the cross-layer protocol DAPR (Distributed Activation with Predetermined Routes) [1] and the sensor network middleware MiLAN (Middleware Linking Applications and Networks) [2]. DAPR is a cross-layer protocol that integrates the sensor selection, routing, and MAC layers. MiLAN is a *proactive* middleware that controls the sensors in order to meet the application quality requirements by adapting the network to the application needs as they vary over time, specifying which sensors should send data, route packets, etc.

In the next sections we describe DAPR and MiLAN in greater detail. Following this, we propose an extension to MiLAN to fully utilize all the advantages provided by a cross-layer design such as DAPR. Then, we provide initial results that show the advantages of using MiLAN to control DAPR. Finally, we provide a discussion of related work and conclusions.

2. DAPR and MiLAN

2.1 DAPR: a Fully Integrated Protocol

DAPR [1] is a cross-layer protocol that quantifies a sensor's importance to the application to avoid using the key sensors as routers in the network. Each sensor is assigned an application cost that reflects the criticality of the node's data, and DAPR selects routes with the smallest cumulative cost to the base station. Simultaneously, nodes whose coverage is not beneficial to the network are turned off to conserve energy.

In DAPR, the data sink sends periodic queries that trigger three phases that form a round, as shown in Figure 1. Each DAPR round starts with a route discovery phase, followed by a role discovery phase, and finally the query processing phase during which routes and node activations are typically fixed.

2.1.1 Route Discovery Phase. All nodes are assigned an application cost. This cost represents the importance of the sensor's data to the application, and is a parameter that can be changed for different applications. For example, for an application that requires full coverage of the network at all times, application cost provides a measure of the sensor's coverage of a certain region relative to the coverage of this region provided by other sensors (see [1] for details).

Routes are set up when the sink floods its *query* packet to indicate an interest in data. These queries are sent via shortest-cost paths, thereby setting up reverse lowest cumulative cost routes to the sink. Once routes are set, each node routes packets through these low cost paths.

2.1.2 Role Discovery Phase. In the next phase, sensors whose data are not beneficial for the application can be turned off without degrading the network's quality of service. For example, for coverage applications, sensors whose data does not add to the global coverage of the network can be turned off, saving their energy for later. The delay to determine whether a node should turn off is proportional to its cumulative application cost so that sensors with high cost routes are turned off first.

2.1.3 Query Processing Phase. During the final phase, all active sensors send their data to the sink through the pre-determined routes for a period of time (V_s) that can be altered depending on application needs. A longer query processing phase means less overhead for the protocol but also less optimal network operation, as active sensors and routes are not updated frequently.

2.2 MiLAN: a Sensor Network Middleware

Several sensor network applications require that only certain sensors be activated at a given time, and that this set of required sensors change over time. For example, a home/office security application may require that only entries to a room be monitored during normal operation mode, whereas different sensors are needed when an intrusion is detected. Similarly, a health monitoring system may require only a few vital signs to be monitored while the individual is healthy, but when signs of stress are detected, more vital sign monitoring is needed. Controlling sensors for such dynamic operation, where the set of active sensors depends not only on the sensors themselves (remaining energy, location, etc.) but also on the state of the application and the system being monitored, is difficult to do directly from the application. MiLAN was designed to ease this burden on the application designer by incorporating all the neces-

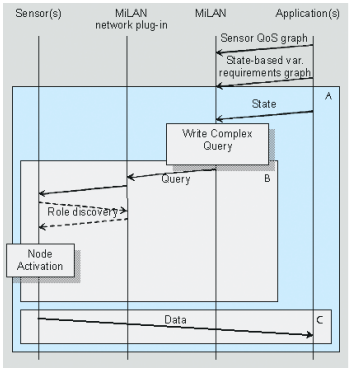

Figure 2. Overview of MiLAN. A and B start when the application starts or changes its state depending on data received from the sensors. C is the normal mode of operation when conveying the sensors data to the application.

sary mechanisms for controlling the network within the middleware [2]. MiLAN provides the application an API through which it can specify its quality of service (QoS) goals, and MiLAN takes care of appropriately setting network parameters to meet these QoS goals over time. Large gains in lifetime can be obtained by allowing MiLAN to adapt a set of tunable network parameters over time to just meet the application's QoS requirements.

We assume that the quality of different sensed data can be quantitatively evaluated. In order to configure the sensors and the network parameters to meet application needs, MiLAN must know (1) the variables of interest to the application, (2) the required QoS for each variable, and (3) the quantitative QoS of data output by sensors. All of these quantities may vary over time. This information is conveyed to MiLAN through "State-based Variable Requirements" and "Sensor QoS" graphs, as shown in Figure 2 (see [2] for details). Using this information, MiLAN configures the sensors and begins the normal network operation, where sensors send data to the application.

The following section discusses how to use MiLAN to control the operation of DAPR, thereby improving overall lifetime of the network and easing the application of the burden of sensor and network management.

3. Managing DAPR Through MiLAN

Previous work on MiLAN focused solely on one-hop networks using standard, layered protocol architectures (e.g., Bluetooth, IEEE 802.11). Multi-hop heterogeneous networks and cross-layer architectures offer

new challenges to MiLAN. We believe that MiLAN can exploit the tunable parameters of a cross-layer protocol like DAPR to save energy while meeting application QoS for such multi-hop networks.

3.1 Overview of MiLAN/ DAPR Combination

Using MiLAN to orchestrate a network running DAPR will provide many advantages in terms of extending network lifetime while meeting dynamic QoS constraints. Parameters of DAPR such as application cost definition and query interval can be adapted to provide maximum benefit to the application. To effectively manage DAPR, MiLAN must obtain the application's current QoS requirements as well as the system state. This global view of the application is used to create queries through which MiLAN communicates with DAPR, as depicted in Figure 3.

3.1.1 Complex Queries. MiLAN has a clear view of the application's required QoS at every point in time. Its main goal is thus to issue queries understandable by DAPR to meet the QoS goal while minimizing energy dissipation. However, new and more sophisticated applications will require *complex* queries. Such information as the monitored variables, the required precision (also called confidence) for each variable of interest, the area of interest in the network, and the reporting mode (*continuous* or *discrete*) need to be included in these queries.

3.1.2 Variables and Precisions. Variables correspond to physical attributes (e.g., temperature, intruder presence) that can be described by XML tags. Each node should know which variable it can monitor and with what precision prior to deployment.

MiLAN also has global information on potential or future application requirements. For instance, it may be readily known to MiLAN that a precision of $conf(v_j)$ on variable v_j is never needed. We propose using a vector P^j of weights or probabilities inferred from the various graphs inherited from the application that indicates the relative importance of each precision $conf(v_j)$ for variable v_j. For simplicity, we quantize $conf(v_j)$ by 10%, and thus P^j indicates how often the variable v_j is required in different precisions from 0.0 to 1.0 in steps of 0.1.

In addition, MiLAN's knowledge of the potential QoS requirements can lead to the conclusion that a variable will never be needed, or that a variable is extremely important in all application states. Thus, MiLAN also issues a vector W that weights all the variables in the system. W and P help assess the relative importance of all sensors to aim at conserving the most critical ones.

3.1.3 Entities within the Monitored Region. The network can also be divided into entities or groups of sensors. An entity can be described by an XML tag such as *windows*, *soldier A*, or *second floor*. We contend that each sensor needs to know to which group of sensors it belongs, using high-level semantic descriptions. For practical reasons, we propose that all sensors are attributed a tag, that is shared with other sensors monitoring the same entity. MiLAN can use this tag to specify the entity of interest in the queries.

3.1.4 Setting the Query Interval. The query interval is a critical factor in the protocol overhead, and thus in the network lifetime. While an application might need to frequently change the active sensors because of changing QoS requirements, a small query interval degrades the network lifetime. Moreover, for an application that requires full network coverage, while a frequent query update can preserve 100% coverage for longer times, it leads to a quicker mid to low range coverage degradation later in time.

Intuitively, we can see that application states that correspond to slowly changing situations probably do not require frequent network changes and thus long query intervals are desirable. Once a more critical state is reached, such as a high stress state in a medical application, a smaller query interval is likely to be used to meet possibly rapidly changing application QoS. On the other hand, if the maximum attainable QoS is already significantly degraded, the query interval should be made longer to increase the network lifetime by lowering the overhead.

3.2 Routing: New Variable-based Cost

For complex, non-coverage-type applications that require the network to monitor one or more variables, any group of sensors that provides data about each variable with greater than the required precision (confidence) provides 100% utility or QoS. In some cases, only one sensor's data may be enough to provide 100% QoS. For these new variable-based applications, the application cost in DAPR should evaluate the relative importance of a node with regard to each variable. Thus, for this new variable-based QoS metric, we define application cost for sensor k as:

$$C_k = \sum_{j=1}^{V} W_j \cdot Cv_{j,k} \tag{1}$$

where V is the number of variables the network is capable of monitoring, W_j is the weight of variable v_j and the per-variable cost is:

$$Cv_{j,k} = \frac{[\sum_{i=1}^{10} P_i^j \text{ such that } \text{conf}(v_{j,k}) \geq P_i^j] \cdot \frac{1}{E(k)}}{\sum_{m=1}^{N} [\sum_{i=1}^{10} P_i^j \text{ such that } \text{conf}(v_{j,m}) \geq P_i^j] \cdot \frac{1}{E(m)}} \quad (2)$$

where P^j is the probability vector (P_i^j is the i^{th} element of P^j), *conf*($v_{j,k}$) is the confidence with which sensor k can monitor variable v_j, and $E(k)$ is the remaining energy of sensor k. C_k evaluates the attention given to each variable and divides the precision of sensor k by the sum of all precisions across all the sensors in the monitored entity. The normalized inverse of the remaining energy is included to differentiate between nodes whose remaining energy is very low but are critical to the application and other sensors whose energy is still high. The sum in the denominator is not an algebraic sum of all precisions *per se*, but rather an evaluation of all the energy devoted to monitoring a variable with various precisions or confidences.

3.3 Node Activation: a Distributed Process

New challenges in a multi-hop network, as well as the specificities of DAPR, require that the node activation process be distributed. DAPR's node deactivation process ensures that a node will deactivate only if the application requirements are met by other active sensors; otherwise, if its own precision is within the tolerance of the required precision and no other lower cost sensor can provide this data, the node will stay active. Nodes consider deactivating in the order inverse to their cost. Figure 3 illustrates this process.

Immediately after this procedure, the selected nodes send a data packet to the sink using their pre-computed smallest cumulative cost path. Only the activated sensors and those contributing to the return path (those that retransmit the packet) will remain active in the round.

4. Preliminary Results

This section presents results that will show the merits of using MiLAN to manage DAPR over a simpler architecture that aims to serve a fixed (often the strictest) QoS. The simulations are carried out in Matlab and use one-hop networks to determine the advantage of intelligent selection of the active sensors based on varying QoS requirements. A fixed amount of energy per second is used by the active nodes (0.1% of the total initial energy).

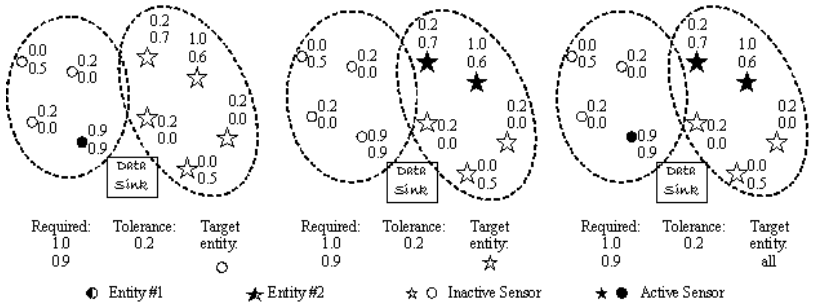

Figure 3. Nine sensors are monitoring two different entities (e.g., two different soldiers in range of one another). The required precisions for the two variables of interest are 1.0 and 0.9, with a tolerance of 0.2. Three cases are possible: only the first entity (with sensors represented by circles), only the second entity (with sensors represented by stars) or both entities are of interest.

4.1 One Variable with Low QoS Requirements

We first compare our application cost to the inverse of the energy cost in order to validate our choice of application cost. For this simulation, three sensors, each with the same initial energy, are able to monitor one variable with precisions 0.1, 0.5, and 0.9 for sensors 1, 2, and 3, respectively. The only QoS requirement is a non-zero precision, i.e., some idea of the measured variable is all that is needed.

Figure 4 shows the remaining energy of each node using application cost and the inverse of remaining energy cost for a given P vector. The sensors' energy decreases by the same amount when the cost is the inverse of the energy since all nodes are eligible to be picked for activation, and thus they are selected in a round-robin manner. Conversely, application cost allows the energy of the most important sensor (with QoS 0.9) to be spared for longer. When sensor 1 dies, sensors 2 and 3 have half and full remaining energies; when sensor 2 dies, sensor 3 has a third of its energy remaining. Application cost would allow for better QoS as illustrated by the shaded area, were the application to request it.

The effect of P is not trivial: when a higher weight is assigned to high precisions, node 3 is spared for longer than when all weights are the same, at the disadvantage of the remaining sensors. It may be good to conserve the energy of important sensors, but this could be unnecessary in some applications where high precision is never needed. We believe statistical information on the application can help find a suitable P.

These results show the merits of our application cost. However, both application cost and the inverse of energy cost bring a significant im-

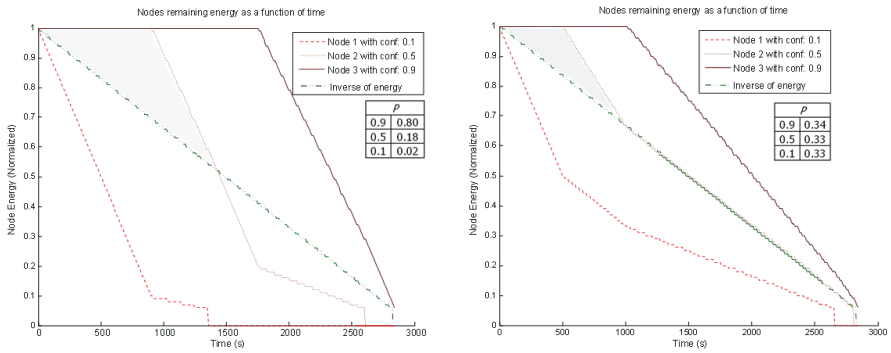

Figure 4. Node remaining energy over time. All three nodes are able to monitor a single variable with precisions 0.1, 0.5, and 0.9. The QoS requirement is 0.1. a) and b) have different P vectors.

provement compared to a system architecture that does not use a middleware to manage the network. To understand the full improvements of our design, we need to evaluate the lifetime and QoS for a network that does not have the benefit of a middleware, and thus simply selects sensors providing the highest QoS available in the network.

4.2 QoS Increase with a Middleware

The conditions of these simulations are similar to the previous experiments. We compare our system design (MiLAN managing DAPR) to a system that has no smart middleware. The latter architecture is only able to activate nodes that will provide the highest QoS available in the network and does not adapt the network QoS to the changing application needs. To quantify the gains in QoS, we calculate an index that averages the percentage of requirement satisfaction. For instance, if the application requires a precision of 0.9 but the network can only provide 0.5, the index is equal to $^{0.5}/_{0.9}$. Obviously, this QoS index has no concrete significance since the precisions are not linear; however, we believe it provides a good basis for comparison.

Figure 5 shows this QoS index during the simulation time (top graph). MiLAN leads to significant improvements by adapting the network QoS to that of the application (bottom graph). On the other hand, the simpler design that selects the sensor with the highest precision at all times uses the energy of high precision or scarce nodes first, even when this is not necessary. During the simulation time, our architecture is always able to provide nominal QoS, while the system with no middleware support offers poor precision during many of the critical stages where a maximum

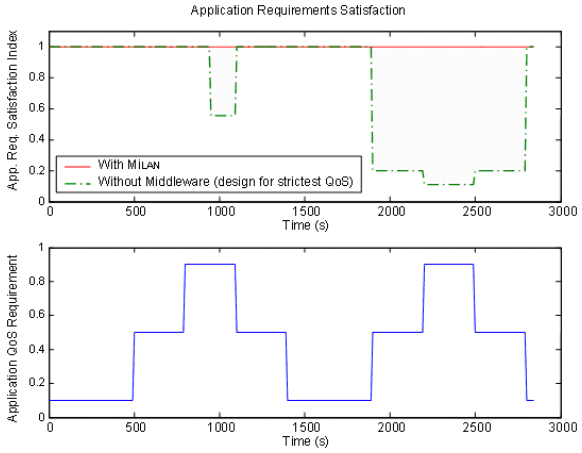

Figure 5. A measure of the QoS differences between a system with MiLAN managing DAPR and a system designed for the strictest QoS. The graph at the bottom shows the application QoS requirement.

QoS is needed. The shaded areas represent the QoS gains ascribable to MiLAN.

4.3 Change of Query Interval

Figure 6 presents a comparison of simulations with an adaptive and a fixed query interval from the perspective of QoS delivery. In these simulations, two sensors are able to monitor a variable with precisions 0.5 and 0.9. The application requires precisions of 0.5, 0.9, or has no interest in the variable, as shown in Figure 6 (c). In the adaptive query interval scheme, the QoS provided by the network is submitted to "gravity": when the application QoS requirement increases, MiLAN waits for it to be constant for 30s (a "weight") before it issues a new query. Conversely, when the QoS requirement decreases, MiLAN sends a query immediately. In the fixed scheme, the query interval is set to 100s. Every 100s, MiLAN checks for the current application requirement and floods a query with this information. The network will try to meet the QoS for the remaining query interval. In both cases, flooding a query costs the same amount of energy, 0.5% of the initial energy in our simulations.

Figure 6 (a) and (b) present the QoS provided by the network in both schemes. Figure 6 (b) shows that much of the energy is wasted when the query is issued at the time of a QoS application requirement peak. The network can also miss QoS requirements for up to 90s. In contrast, an adaptive query length design is able to respond to an increase in the

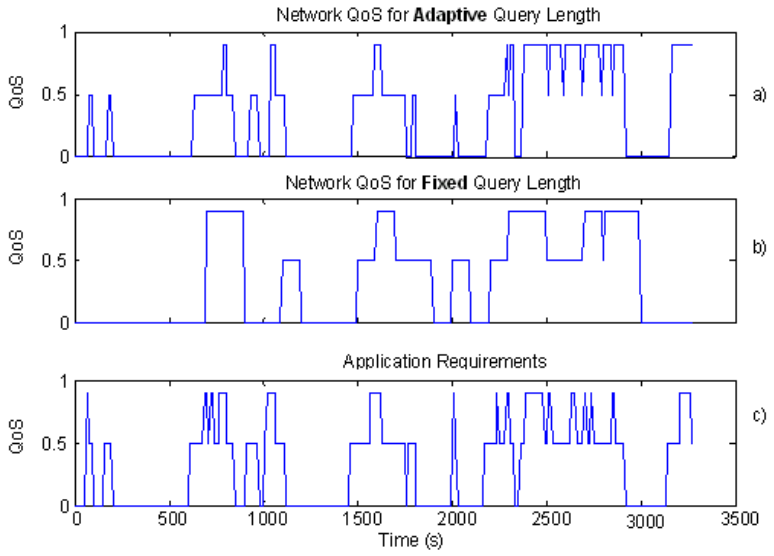

Figure 6. Network QoS response of the adaptive (a) and fixed (b) query length schemes to the application needs (c).

application need within its 30s weight. Consequently, the QoS index for the adaptive query length scheme is 40% higher than that of the fixed query interval scheme.

Some applications may need an immediate response from the network. We simulated an adaptive query length scheme with no gravitational pull and found that the application requirements are always met, but for a shorter time—the network lifetime is reduced by about 10%.

5. Related Work

K. Römer et al. [3] offer a look at challenges in designing middleware for sensor networks. They identify the need for data-centric communications, adaptive fidelity functions, and QoS knowledge. They also define middleware tasks as assigning high-level and complex sensing formulation and adaptation to the network. Our research mostly follows the precepts delineated by K. Römer et al.

Y. Yu et al. present a cluster-based middleware in [4] and provide a level of abstraction that separates application semantics from the layered protocol architecture. They introduce a cluster forming layer (within the middleware virtual machine) responsible for building clusters around a phenomenon in a distributed manner. They define *knobs* or mechanisms

designed to change the working mode of a sensor. These knobs are similar in spirit to the tunable parameters presented in our work.

In [5], Cornea et al. study middleware optimizations for cross-layer architectures dedicated to interactive mobile video streaming. The middleware identifies the viewing device and forwards this information to the nodes, who then adapt their sleep and active periods, and their bit rate, frame rate, and video resolution to the needs of the viewing device.

To the best of our knowledge, our work is the first to study the details of using a middleware to manage a cross-layer architecture and to prove the advantages of constantly adapting network and sensor parameters, reflecting the application's need to optimize the underlying structures.

6. Conclusions

We have shown the advantages of using a middleware to supervise a cross-layer protocol: the combination of MiLAN and DAPR is able to continuously adapt the sensor network to the application QoS requirements. Gains in network lifetime and QoS are substantial when the system design spares sensors whose data is critical to the application. We have also shown the advantages of using MiLAN to adapt the QoS requirements over a design with non-adaptive QoS. Finally, MiLAN takes full advantage of DAPR's flexibility by adjusting its tunable parameters such as complex queries, query interval, and application cost. Through this tight cooperation, MiLAN is able to effectively manage the underlying DAPR protocol while providing a level of abstraction to the application programmer.

References

[1] M. Perillo and W. Heinzelman, "DAPR: A protocol for wireless sensor networks utilizing an application-based routing cost," in *Proc. of the IEEE WCNC*, 2004.

[2] W. Heinzelman, A. Murphy, H. Carvalho, and M. Perillo, "Middleware to support sensor network applications," *IEEE Network*, vol. 18, pp. 6–14, Jan. 2004.

[3] K. Romer, O. Kasten, and F. Mattern, "Middleware challenges for wireless sensor networks," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 6, no. 2, 2002.

[4] Y. Yu, B. Krishnamachari, and V. K. Prasanna, "Issues in designing middleware for wireless sensor networks," *IEEE Network Magazine*, vol. 18, Jan. 2004.

[5] R. Cornea, S. Mohapatra, N. Dutt, A. Nicolau, and N. Venkatasubramanian, "Managing cross-layer constraints for interactive mobile multimedia," in *In Proc. of the IEEE Workshop on Constraint-Aware Embedded Software*, 2003.